Design and Analysis of Algorithms Assignment 3

Harrison Lee, Alex Zhao

February 7, 2019

Question 1. *(6.5) String Segmentation (Segmented Least Squares Fit? As from class.)*

Given: Given a string of letters $y = y_1y_2...y_n$ with a length of n. We can assume we can get the quality of any string using the function $quality(x)$ for string x. Total quality of a segmentation is the qualities of each of its blocks.

Find: Find an efficient algorithm the gets the segmentation of maximum total quality.

Algorithm 1. *Find the optimal string segmentation.*

Proof. Subproblems: $OPT(j)$, is the total quality of the optimal segmenation from character 1 to character j. $Segment(i, j)$ is the segmentation from i to j.

Recurrence: $OPT(j) = max\{OPT(i) + quality(Segment(i + 1, j))$ for all i from 1 to $j - 1$

This is similar to the answer for Segmented Least Squares Fit, as discussed in lecture. The main change needed is to replace the error function with the quality function in this problem and to maximize quality rather than minimize error.

Full Algorithm:

$OPT(1) = quality(x_1)$ // Base Case

$segmentations$ = [[1]] // List of list of segmentation break points for each $OPT(j)$

for j = 2, j $\leq$ n, j++

  currentQuality = $quality(Segment(1, j)$

  currentSegmentation = 1

  for i = 2, i ¡ j, i++ // Find $OPT(j)$

    if $OPT(i)$ + $quality(Segment(i + 1, j) \geq$ currentQuality

      currentQuality = $OPT(i)$ + $quality(Segment(i + 1, j)$

      currentSegmentation = i

  $OPT(j)$ = currentQuality

  $segmentations[j] = segmentations[i] + currentSegmentation$

return $OPT[n], segmentations$

Running Time: $O(n^2)$. Each character is compared to the OPT values of all prior characters.

□

Question 2. *(6.28)*

a. *Every job in J can be completed before its deadline by definition, meaning there is a schedule where the max lateness is 0 Ordering by increasing deadline produces a schedulable set that minimizes max lateness (proven in class). Using this order will create a schedule that has max lateness 0 (we know that is the minimum max lateness). Therefore, there is a schedule for J where the jobs are ordered by increasing deadline.*

b. *Let the jobs be ordered by increasing deadline (we know J can have this ordering from part a.*

Simple observation:

Given n jobs, the last job j_n is either in J or not in J. If j_n is in J, then the optimal solution is the optimal solution when given all the earlier jobs and a new maximum deadline D of the start time of j_n (either $D - t_n$ or $d_n - t_n$, whichever is earlier). If j_n is not in J, then the optimal solution is just the optimal solution given all the earlier jobs and the same max deadline.

Subproblems:

$OPT[i, d]$ is the optimum solution given jobs from 1 to i where no job can run past deadline d.

Recurrence:

$$OPT[i,d] = max \begin{cases} OPT[i-1,d] \\ OPT[i-1, min(d, d_n) - t_i] + 1 \end{cases}$$

Either you don't take j_i in which case J doesn't get any bigger and you look at the subproblem without j_i and the same max deadline, or you take j_i in which case J becomes 1 bigger and you look at the subproblem without j_i and a the new earlier max deadline.

Full Algorithm:

$OPT[0, d] = 0, OPT[i, 0] = 0$ (can't have any jobs in J without any jobs, or without any time)

You can fill in the table in any order (row by row, column by column, diagonally) and you can only choose to take j_i if d_i does not exceed d.

The final solution is $OPT[n, d_n]$.

The runtime for this algorithm is $O(n \cdot d)$ (there are $n \cdot d$ subproblems and each subproblem looks at a constant number of subproblems.